# Decorators

## Summary

- Decorators are often used in frameworks (eg Angular, Vue) to chance and enhance classes and how they behave.
- We can apply decorators on classes, properties, methods, parameters, and accessors (getters and setters).
- A decorator is just a function that gets called by the JavaScript runtime. In that function, we have a chance to modify a class and its members.
- To use decorators, we have to enable the **experimentalDecorators** setting in tsconfig.
- We can apply more than one decorator to a class or its members. Multiple decorators are applied in the reverse order.

 

# Cheat Sheet

### Class decorators

```
function Component(constructor: Function) {
  // Here we have a chance to modify members of
  // the target class.
  constructor.prototype.uniqueId = Date.now();
}

@Component
class ProfileComponent { }
```

### Parameterized decorators

```
function Component(value: number) {
  return (constructor: Function) => {
    // Here we have a chance to modify members of
    // the target class.
    constructor.prototype.uniqueId = Date.now();
  };
}

@Component(1)
class ProfileComponent {}
```

### Decorator composition

```
// Multiple decorators are applied in reverse order.
// Pipe followed by Component.
@Component
@Pipe
class ProfileComponent {}
```

 

### Method decorators

```
function Log(target: any, methodName: string, descriptor:
PropertyDescriptor) {
  // We get a reference to the original method
  const original = descriptor.value as Function;
  // Then, we redefine the method
  descriptor.value = function(...args: any) {
    // We have a chance to do something first
    console.log('Before');
    // Then, we call the original method
    original.call(this, ...args);
    // And we have a chance to do something after
    console.log('After');
  }
}

class Person {
  @Log
  say(message: string) {}
}
```

### Accessor decorators

```
function Capitalize(target: any, methodName: string, descriptor:
PropertyDescriptor) {
  const original = descriptor.get;
  descriptor.get = function() {
    const result = original.call(this);
    return 'newResult';
  }
}

class Person {
  @Capitalize
  get fullName() {}
}
```

 

**Property decorators**

```
function MinLength(length: number) {
  return (target: any, propertyName: string) => {
    // We use this variable to hold the value behind the
    // target property.
    let value: string;

    // We create a descriptor for the target property.
    const descriptor: PropertyDescriptor = {
      // We're defining the setter for the target property.
      set(newValue: string) {
        if (newValue.length < length)
          throw new Error();
        value = newValue;
      }
    }

    // And finally, we redefine the property.
    Object.defineProperty(target, propertyName, descriptor);
  }
}

class User {
  @MinLength(4)
  password: string;
}
```

 